UCF Stands For Opportunity

CDA6530: Performance Models of Computers and Networks

Chapter 8: Discrete Event Simulation Example --- Three callers problem in homwork 2

SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE

Problem Description

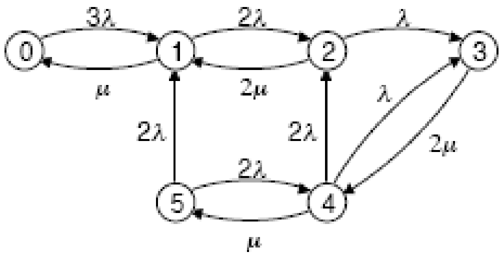
- Two lines services three callers. Each caller makes calls that are exponentially distributed in length, with mean $1/\mu$. If both lines are in service by two callers and the third one requests service, the third caller will be blocked. A caller whose previous attempt to make a call was successful has an exponentially distributed time before attempting the next call, with rate λ. A caller whose previous call attempt was blocked is impatient and tries to call again at twice that rate (2λ), also according to exponential distribution. The callers make their calls independent of one another.

Analysis Results

Define the following six states:

0 no calls in progress, 3 callers idle
1 1 call in progress, 2 callers idle
2 2 calls in progress, 1 caller idle
3 2 calls in progress, 1 caller impatient
4 1 call in progress, 1 caller impatient
5 0 calls in progress, 1 caller impatient

The state transition diagram is



The rate generator matrix is

$$\underline{\underline{Q}} = \begin{bmatrix} -3\lambda & 3\lambda & 0 & 0 & 0 & 0 \\ \mu & -\mu-2\lambda & 2\lambda & 0 & 0 & 0 \\ 0 & 2\mu & -2\mu-\lambda & \lambda & 0 & 0 \\ 0 & 0 & 0 & -2\mu & 2\mu & 0 \\ 0 & 0 & 2\lambda & \lambda & -\mu-3\lambda & \mu \\ 0 & 2\lambda & 0 & 0 & 2\lambda & -4\lambda \end{bmatrix}$$

- Steady state prob: π

$$\pi \mathbf{Q} = 0$$

$$\pi \mathbf{1} = 1$$

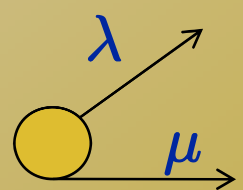
- Matlab code:

```
Q = [………];
Pi = zeros(1, 6);
Q_m = [Q(:, 1:5)  ones(6,1)];
B = [0 0 0 0 0 1];
Pi = B * inv(Q_m);
```

Simulation based on Markov Model

Pre Simulation

- Strictly refer to the state transition diagram
  - Remember current state: currentState
  - Determine next state: nextState
- This is a continuous-time Markov Chain
  - State duration time:
    - Exp. distr. with rate ($\lambda + \mu$)
    - Determine the next transition event time
  - At the time of transition event:
    - Use discrete r.v. simulation method to determine nextState:
      - Transit first path with prob. of $\lambda/(\lambda+\mu)$
      - Transit second path with prob. of $\mu/(\lambda+\mu)$



Pre Simulation

- Events:
  - Transition out from currentState to nextState
- Event List:
  - EL = t_{tran}: time of the next transition event
  - Simpler than queuing systems
- Output:
  - Tran(i): event time of the i-th transition
  - State(i): system's state after i-th transition
- Termination condition:
  - N: # of transitions we simulate

Simulation

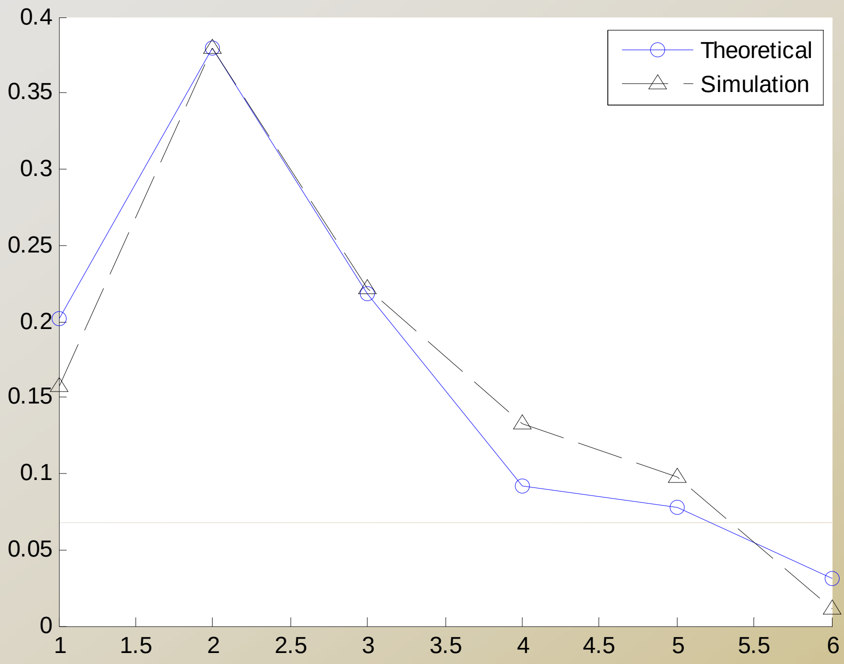
```
Set stateN, initState, N, lambda, mu, Q
currentState = initState; currentTime = 0;
for i=1:N,    % simulate N transitions
     % first, simulation currentState during time (next event time)
     % Given that we know the Markov model and the Q matrix
     outRate =  - Q(currentState, currentState);
     Tran(i) = currentTime - log(rand)/outRate; % exp. distr. with rate of outRate
     % next, determine which state transits to?
     U = rand;
     vector = Q(currentState,:); vector(currentState) = 0;
     for j=1:stateN,
         if U <= sum(vector(1:j))/sum(vector),
           nextState = j; break;
         end
     end
     State(i) = nextState;
     currentState = nextState; currentTime = Tran(i); % prepare for next round
end
```

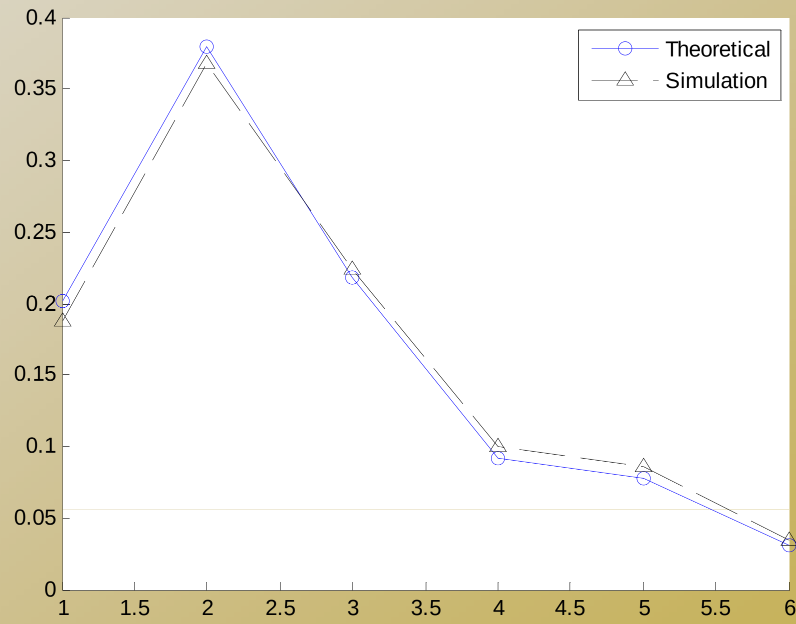
Post Simulation Analysis

- Objective:
  - Compute Pi based on simulation
- Pi(k) = time spent in state k / overall simulation time
  - Overall simulation time = Tran(N)
  - Time spent in state k: Time(k)

```
Time = zeros(6,1); Time(initState) = Tran(1);
for k=1:6,
    for i=1:N-1,
        if State(i) == k,
            Time(k) = Time(k) + Tran(i+1) - Tran(i);
        end
    end
end
```

Simulation Results





N=100

N=5000

- Shows that our simulation is consistent with analytical result

Realistic Simulation With physical meaning

Problem for the Simulation Above

- The simulation actually simulates continuous-time Markov Chain only
  - Only based on Markov model
  - The simulation does not really simulate the physical world events
    - Three callers? What's their status?
    - Two service lines?
- More accurate & realistic simulation
  - Simulate the physical entities actions/behaviors/events

Pre Simulation

- What physical entities should we consider?
  - Should directly correspond to physical entities
  - Should uniquely define system status
- There are two types of entities
  - Two service lines
  - Three callers
- If we do not care which service line is working
  - We should treat three callers as simulation nodes

Pre Simulation

- Each caller's data:
  - status: 'patient', 'impatient', 'calling'
    - Caller = ['P' 'I' 'C'];
  - nextT: time tick for its next action
    - Finishing phone call
      - When current status is 'calling'
    - Making phone call attempt
      - When current status is 'idle' or 'impatient'
- Event list:
  - Each caller only has one next event/action
  - Event list: EventList[3]
    - Three nodes' next action time
    - We do not really need to save nextT in caller data

Pre Simulation

- Next event: the smallest time in EventList
  - Suppose it is EventList[k]
  - Update system at this time EventList[k]
    - Move simulation time to this time
    - Check caller k: what's its action?
    - Simulate the next event time nextT for caller k
      - Based on its next status: calling? Patient? Impatient?
      - We need to know the status of those two service lines in order to determine this
        - serveLineNum: # of lines that are using
    - Update EventList[k] = nextT

Pre Simulation

- Update output data:
  - Tran(i) = EventList[k]
  - State(i): system's state after this node action
    - In order to compare with analytical results
  - If we care about each caller's behavior:
    - Tran(i) = EventList[k]
    - ActCaller(i) = k
      - The k-th caller acts at time Tran(i)
    - CallerState(i) = Caller(k)
      - k-th caller's state after the i-th event
      - The other callers do not change their state after this event

Simulation Pseudo Code

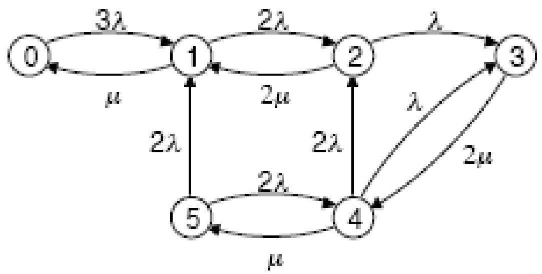
```
Initialize N, \lambda, \mu, State[], Tran[]
Initialize initState and Caller[3]; currentTime = 0;
Initialize EventList[]  (use corresponding distribution to generate)
For i=1:N,
        Find the smallest time tick in Eventlist[] → index is k
   % caller k's action is the event we simulate now
        currentTime = EventList[k];
        Update caller k's  status;
        Update how many phone lines are used
        Generate caller k's next action time, assign to EventList[k]
   % Update output data
        Tran(i) = currentTime;
        State(i) = ?  (case statement to decide based on state definition)
End
```

- State(i) = ? (case statement to decide based on state definition)
- E.g.:
  - [C,C,I] → state 3
  - [I,C,C] → state 3
  - [P,C,I] → state 4
  - …

Define the following six states:

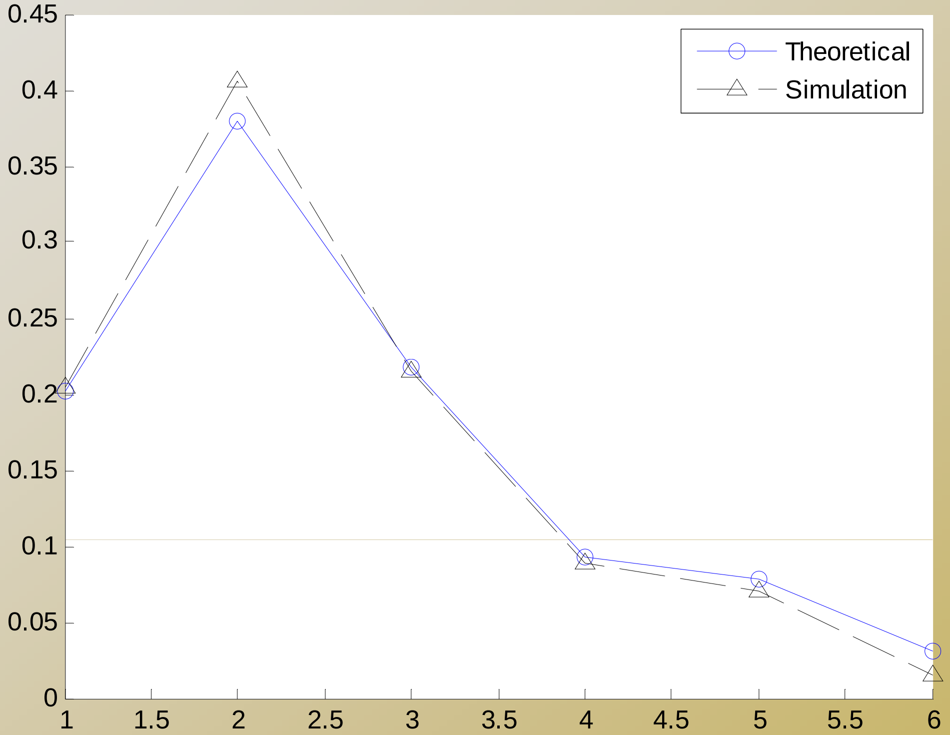
0 no calls in progress, 3 callers idle
1 1 call in progress, 2 callers idle
2 2 calls in progress, 1 caller idle
3 2 calls in progress, 1 caller impatient
4 1 call in progress, 1 caller impatient
5 0 calls in progress, 1 caller impatient

The state transition diagram is



The rate generator matrix is

Simulation Compared with Analysis



N=1000

Conclusion

- The realistic simulation uses minimal amount of knowledge of statistical analysis
- Realistic simulation directly simulate real world entities actions and behaviors
- The model-based simulation is still useful
  - Better than no simulation
  - Applicable for all systems described by one model
  - Can study system's performance when there is no analytical results
  - Sometime realistic simulation is too complicated or take too long to do
- We need to decide which simulation to conduct